

# Rapport Projet Langage C



## **Convertisseur Système numérique Égyptien**

Du 18/12/2017 au 9/02/ 2018

Clément LE BAS - Haseeb JAVAID - Thomas BLANCO - Mathieu JUGI

# Sommaire

## 1.Introduction

1.1 Contexte du projet.....	page 3
1.2 État de l'art.....	page 4
1.3 Pourquoi ce choix ?.....	page 14

## 2. Description

2.1 Algorithme .....	page 15
2.2 Limitation théorique .....	page 17
2.3 Difficultés .....	page 18

## 3. Présentation du code

3.1 Choix des types de données .....	page 19
3.2 Difficultés techniques .....	page 19
3.3 Choix de résolution des difficultés .....	page 20
3.4 Morceaux de code .....	page 22

## 4. Conclusion

4.1 Technique .....	page 24
4.2 Générale .....	page 24
4.3 Ouverture possible .....	page 24

Annexes .....	page 26
---------------	---------

Webographie .....	page 28
-------------------	---------

# Introduction

## 1.1 Contexte du projet

Le but de ce projet est de permettre d'expliquer plusieurs systèmes de numérations existants ou ayant existés, et cela à travers la réalisation d'un convertisseur en C.

Pour cela, nous avons formé différents groupes(5 grp de 4), avec pour chaque groupe une tâche à accomplir pour aboutir à la création du convertisseur. En effet, chaque groupe choisit un système de numération particuliers et réalise une fonction en C permettant de convertir un nombre decimale dans le système de numération choisit. De plus, la création d'un manuel d'utilisation est à prévoir pour chaque groupe.

Un groupe à été désigné pour réaliser le main (programme principal ) donnant un menu proposant les différentes systèmes de numérations à l'utilisateur. Ce groupe gère donc l'appel des différentes fonctions mais aussi les limites des systèmes de numération décrits dans le manuel d'utilisation (ex : ne peut pas aller au delà de 9999). Evidemment, ce groupe doit lui aussi créer une fonction de conversion comme pour les autres groupes.

Ainsi, les groupes ont été formés selon le niveau de chacun en programmation C afin de permettre une réalisation optimale du projet et une recherche bibliographique a été effectuée au préalable pour chaque groupe afin de trouver un système de numération pour chacun, mais aussi pour voir les systèmes de conversions déjà existants, c'est ce que l'on va voir dans la partie suivante.

## 1.2 État de l'art

### Les convertisseurs :

À la manière du projet que nous sommes en train de réaliser, il existe d'autres technologies/convertisseurs du même genre. La plupart de ceux sont codés en Javascript.

Voici un exemple type de ce que l'on peut trouver :

The image shows a web-based number conversion tool. On the left, a panel titled "Selectionnez des systèmes de numération" (Select numbering systems) contains a list of options with checkboxes: "chiffres romains" (checked), "français en toutes lettres" (unchecked), "numération maya" (checked), "numération égyptienne" (checked), "système binaire" (checked), "système hexadécimal" (unchecked), and "système octal" (unchecked). On the right, the tool displays the conversion of the number 5. At the top, the input "5" is shown in a box. Below it, four rows show the results for different systems: "5 en chiffres romains" resulting in "V", "5 en numération maya" resulting in a single horizontal bar, "5 en numération égyptienne" resulting in three vertical bars above two vertical bars, and "5 en système binaire" resulting in "101".

Système de numération	Résultat
chiffres romains	V
numération maya	—
numération égyptienne	 
système binaire	101

*(Lien du site dans la webographie)*

On peut voir ici que le programme gère les erreurs d'entrées.

The screenshot shows a web application for converting numbers between different systems. On the left, a panel titled "Selectionnez des systèmes de numération" (Select numbering systems) contains several checkboxes: "chiffres romains" (checked), "français en toutes lettres" (unchecked), "numération maya" (checked), "numération égyptienne" (checked), "système binaire" (checked), "système hexadécimal" (unchecked), and "système octal" (unchecked). On the right, there is a text input field containing "5.1". Below this input, there are four conversion buttons, each with a yellow header and a white body. The headers are "5.1 en chiffres romains", "5.1 en numération maya", "5.1 en numération égyptienne", and "5.1 en système binaire". Each button's body contains the text "entrez un nombre valide" (enter a valid number), indicating that the input "5.1" is not accepted for these specific conversion targets.

Les nombres à virgules n'existent pas en romain ou en égyptien. Néanmoins, ce n'est pas le cas pour le système binaire. En effet, on peut écrire des nombres à virgules en binaire, en hexadécimal ou dans les autres bases.

Le programme ici admet certaines limites. En plus de ne pas prendre en compte les nombres à virgules, il ne prend pas en compte les nombres inférieurs à -999999999999 ou supérieurs à 999999999999.

Voici un autre exemple de convertisseur qui se concentre sur les

conversions de bases :

### Conversion de Base 10 vers Base N

★ NOMBRE EN BASE 10

★ CONVERTIR EN BASE

### Résultats

[1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1]<sub>(2)</sub>

Conversion en Base N - [dCode](#)

Catégorie(s) : Mathématiques, Arithmétique

L'avantage par rapport à celui vu précédemment c'est que celui ci va de la base 2 à la base 62. De plus, il permet la conversion de nombres à virgule.

Le seul problème est l'interface graphique qui permet à l'utilisateur de mettre des lettres contrairement à l'exemple précédent.

### Conversion de Base 10 vers Base N

★ NOMBRE EN BASE 10

★ CONVERTIR EN BASE

### Résultats

[0]<sub>(2)</sub>

Conversion en Base N - [dCode](#)

Catégorie(s) : Mathématiques, Arithmétique

dCode et vous

(Ici, l'interface graphique n'affiche aucun message d'erreur à l'utilisateur pour lui montrer son erreur.)

## **Les systèmes de numération**

Un système de numération est un ensemble de règles qui régissent une, voire plusieurs numérations données.





































Il en existe beaucoup depuis que l'homme sait communiquer, car chaque peuple quelque'il soit créait le sien ( Maya, sumérien, arabe etc).

Les premiers symboles numériques sont apparus en Mésopotamie, où on été retrouvé des tablettes d'argile datant presque de 5000 ans sur lesquelles on retrouve des traces de numération. Même dans des grottes, on peut y voir des dessins pour dénombrer.





Voici quelques exemples de systèmes de numération que notre groupe a étudié et va vous expliquer pour qu'à la fin vous sachiez écrire les nombres à l'époque du Moyen - Âge, de Babylone et de l'Egypte Antique :

## Numération décimale au Moyen Âge :

Au XIIe siècle, les nombres s'écrivaient jusqu'à 9.999 avec le système de numération suivant :

								
1	2	3	4	5	6	7	8	9
								
10	20	30	40	50	60	70	80	90
								
100	200	300	400	500	600	700	800	900
								
1000	2000	3000	4000	5000	6000	7000	8000	9000

exemples :

			
11	12	13	1419

Ici, pour écrire le nombre 11, il suffit de superposer le signe représentant 1 et celui représentant 10. Ce système de numération est en effet basé sur la superposition de signes. Ainsi, pour obtenir 1419, il a fallu superposer le signe représentant 1000 avec celui du 400, du 10 puis du 9.

Tout d'abord, il faut savoir que chaque nombre de ce système tourne autour de la barre centrale :



Chaque puissance de 10, a son espace dédié autour de la barre verticale ci-dessus.



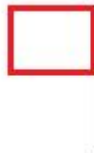
- Les unités :

Pour les unités, l'espace réservé est le coin en haut à droite



- Les dizaines :

Cette fois ci, l'espace se trouve aussi en haut mais à gauche



- Les centaines :

Avec les centaines, on passe en bas et à droite



- Les milliers :

Enfin, il reste un espace de libre, en bas à gauche



Ainsi grâce à cette méthode, lire cette écriture devient beaucoup facile.  
En effet si l'on voit le nombre ci-dessous :



On peut tout de suite deviner que c'est un nombre à un seul chiffre ( 5),  
puisque seulement l'espace réservé aux unités est utilisé.

## Numération Babylonienne :

La civilisation babylonienne a existé au début du deuxième millénaire avant JC, jusqu'au début du premier millénaire en Mésopotamie.

Sa numération est apparue vers l'an 1800 avant JC.

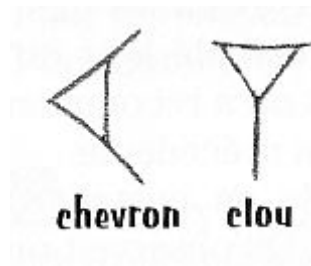
0 (zéro)		1		2		3		4	
5		6		7		8		9	
10		11		12		13		14	
15		16		17		18		19	
20		21		22		23		24	
25		26		27		28		29	
30		31		32		33		34	
35		36		37		38		39	
40		41		42		43		44	
45		46		47		48		49	
50		51		52		53		54	
55		56		57		58		59	

C'est une numération en base 60, telle que nous l'utilisons aujourd'hui pour découper le temps ( 1 heure = 60 minutes ; 1 minute = 60 secondes ) ou encore pour numéroter les angles.

Il n'y avait pas de symbole pour le chiffre 0, jusqu'en -300 avant JC lorsque les Séleucides introduiront un symbole pour cela : . Par la suite, nous utiliserons ce symbole.

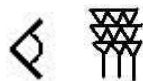
C'est une numération additive de 1 à 59.

Cette numération est seulement composé de deux symboles :






Le clou désigne une puissance de 60 (1, 60, 3600, etc), et le chevron désigne le 10 fois un clou.

Ainsi on pouvait écrire le nombre 19 comme ceci :



- un chevron puis 9 clous.

Au delà de 59, les nombres deviennent positionnels, pour différencier le nombre 60 du nombre 1 c'était difficile, car les deux s'écrivent .

Mais depuis l'ajout du symbole 0, il est beaucoup plus facile de les différencier car 60 s'écrit maintenant  .

Voici un autre exemple d'un nombre plus grand, 5112 :

Il faut commencer par décomposer ce nombre en base 60,

$$5112 = 3600 \times 1 + 60 \times 25 + 12$$

Maintenant, pour son écriture, il faut retenir seulement le 1, le 25 et le 12 dans cet ordre précis, puis on les note avec les symboles babyloniens :









On voit bien qu'il y a de droite à gauche un chevron et deux clous pour le nombre 12, deux chevrons et 5 clous pour le nombre 25 (25 x 60) et enfin un clou pour le nombre 3600 (1 x 3600).

### Numération décimale Égyptienne :

La numération Égyptienne est apparue 3000 ans avant J-C. C'est une numération de type additive. Elle consiste à définir des symboles (hiéroglyphe) pour désigner les nombres 1 ; 10 ; 100 ; 1000 ; 10 000 ; 1 000 000. Ainsi, on peut écrire les nombres jusqu'à 9999999 avec ce système de numération.

Tableau du système de numération égyptien

Valeur	Hieroglyph	Représentation
1		Bâton
10		Anse de panier
100		Papyrus
1000		Lotus
10,000		Doigt
100,000		Têtard

1,000,000		Le Dieu Heh
-----------	---	-------------

**Exemple :**  
53      

Dans cet exemple, on voit que pour un soucis de lisibilité, on regroupe les signes. Ainsi, pour représenter le nombre 53, il faut travailler chiffre par chiffre :

- 5 est à la position des dizaines, 10 est représenté par une anse de panier, il faut donc mettre 5 anses de panier.
- 3 est positionné au niveau des unités, une unité est représentée par un baton, il faut donc mettre 3 bâtons.

## 1.3 Pourquoi ce choix ?

### Numération égyptienne/Moyen Âge

Au terme de cette étude, nous avons opté pour le système de numération égyptien plutôt que celle du Moyen Âge qui d'une part, permet de compter beaucoup plus loin que celui du Moyen Âge (9999999 contre 9.999).

D'autre part, la superposition de signes rend la réalisation du convertisseur beaucoup plus complexe, contrairement au regroupement de signes pour la numération égyptienne.

Enfin, la numération égyptienne peut se coder avec les caractères unicode, ce qui n'est pas le cas de la numération du Moyen Âge .

### Numération égyptienne/Babylonienne

Au terme de cette étude, nous avons opté pour le système de numération égyptien plutôt que le Babylonien qui d'une part, est plus facile à lire que le babylonien. En effet, il faut faire des calculs pour lire des nombres babyloniens(en base 60).

D'autre part, la civilisation égyptienne à durée 3 millénaires (contre 1 millénaire pour les babyloniens) et à beaucoup plus marquée l'esprit des gens via leur architecture, les arts égyptiens, leur croyances religieuses etc...



## DESCRIPTION

### 2.1 Algorithme

Pour la réalisation de ce programme, nous avons procédé en plusieurs étapes. L'utilisateur saisit un nombre entier à convertir. Pour effectuer la conversion et l'afficher en respectant les règles de numération égyptienne, l'algorithme doit permettre de décomposer le nombre.

Par l'intermédiaire d'une boucle while, chaque puissance de 10 doit être soustrait à ce nombre. La numération égyptienne est un système additif et non pas un système de position. L'algorithme doit ainsi prendre en compte cette itération.

Exemple: le nombre 231 est composé de 2 centaines, 3 dizaines et 1 unité. La boucle va ainsi soustraire 2 fois 100, 3 fois 10 et 1 fois 1. Grâce à un compteur, la boucle va déterminer le nombre de symboles à imprimer en fonction de son ordre de grandeur.

Afin de gérer au mieux cette décomposition, nous avons décidé de trier les puissances de 10 dans un tableau. Le système de numération égyptien est limité à 7 puissances de 10 (1, 10, 100, 1 000, 10 000, 100 000, 1 000 000), le tableau est donc de taille 7. En parallèle, un deuxième tableau, constitué des hiéroglyphes, est "relié" à ce tableau de décomposition.

Chaque case du tableau `decompose[ ]` est rempli grâce à une boucle avec une condition de supériorité, et un compteur.

Exemple:

Si l'utilisateur saisit 97865:

`decompose[0]=0`                      `decompose[4]=8`

`decompose[6]=5` `decompose[5]=6`

`decompose[1]=0`

`decompose[2]=9`

`decompose[3]=7`

Ce tableau nous permet par la suite d'imprimer correctement les symboles.

La fonction affichage va traiter chaque composante du tableau. Tant que la valeur de `decompose[i]` est supérieur à 3, on affiche trois fois le symbole correspondant. On soustrait 3 à la valeur de `decompose[i]`. Puis on passe à la case suivante du tableau. Si la valeur de `decompose[i]` est inférieure à 3, l'algorithme imprime le nombre de symboles nécessaires, puis soustrait le restant de `decompose[i]`. Lorsque `decompose[i]` vaut 0, on applique 3 espaces .

L'affichage de la conversion s'effectue de gauche à droite, ligne par ligne. Les règles de disposition des symboles de la numération égyptienne étant assez souple, nous avons décidé d'afficher les hiéroglyphes trois par trois, avec un espace supplémentaire entre chaque ordre, pour une meilleure lisibilité.



## 2.2 Limitation théorique

Dans le système de numération égyptien, la première limitation théorique est que l'on ne peut pas aller au delà de 9 999 999 et en dessous de 1.

De plus, la spécificité de ce système est le fait que le zéro n'existe pas.

Exemple: le nombre 203 se note par juxtaposition des 2 signes représentant le nombre 100 et des 3 signes représentant l'unité. L'absence de dizaine se traduisant par l'absence de symbole représentant le nombre 10:

$$203 \rightarrow \text{𐦢𐦢 III}$$

La seconde correspondait à l'affichage des symboles de cette numérotation car celle-ci doit respecter un nombre de ligne maximum, qui est de 3.

Ensuite il y a le regroupement par ordre de grandeur, c'est à dire, les unités, les dizaines, les centaines, etc... Chaque hiéroglyphe est rangé verticalement par deux, trois ou quatre maximum.

Pour des questions esthétiques et d'occupation d'espace, les signes peuvent être superposés ou non mais ce n'est pas une règle, certains symboles hiéroglyphiques étant plus longs ou plus courts que d'autres.

## 2.3 Difficulté:

- Gestion de l'affichage horizontale

Lorsque nous avons commencé à coder le convertisseur, l'affichage était verticale:



Nous avons donc décidé de reprendre le programme à 0 en utilisant la méthode de décomposition. Comme expliquée précédemment, la fonction 'décomposition' place chaque ordre de grandeur dans le tableau, puis on utilise ce tableau pour afficher les symboles case par case et ligne par ligne.

## PRÉSENTATION DU CODE

### 3.1 Choix des types de données

Pour le choix des données nous avons utilisé

- un pointeur de integer pour le tableau 'décompose'
- un pointeur pour le tableau de symboles égyptiens, pour pouvoir transmettre des données à travers les fonctions (tableau dynamique)
- des integer pour les compteurs de boucle
- un wchar pour représenter les symboles égyptiens (Explication dans la partie 3.3)

### 3.2 Difficultés techniques

Nous avons rencontré plusieurs difficultés techniques durant la réalisation de notre programme :

- Stockage et affichage des caractères unicode dans un tableau.
- Cadrage de l'affichage des nombres égyptiens.
- Problème d'affichage verticale dans le terminal (tout était sur une même ligne)
- Problème lorsque l'on avait un 0 dans un nombre
- Afficher l'unicode dans différents terminaux

### 3.3 Choix de résolution des difficultés

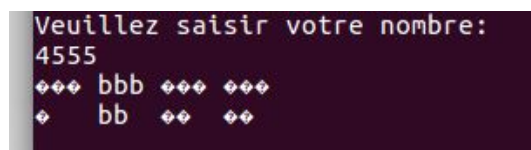
- Gestion de la mémoire et affichage des caractères

Les caractères unicodes nous ont posé des problèmes d’affichage, il n’apparaissaient pas sur le terminal après la compilation. Le type *char* a été conçu pour stocker des caractères qui prennent très peu de place en mémoire. Il ne peut donc stocker que des entiers de -128 à 127. Pour pouvoir utiliser il nous faut un type de caractère plus grand que *char*, qui a une "taille" de 1 octet et ne peut donc stocker que  $2^8$  soit 256 caractères différents.

L'Unicode gère la totalité des caractères présents sur terre. Pour le stocker, le type adapté pour les caractères spéciaux s'appelle *wchar\_t* (Wide Character Type).

Pour cela, nous avons déclaré un tableau dynamique avec le type de variable *wchar\_t*. La syntaxe exige d’ajouter un “L”, devant la valeur lors de l’initialisation. Lors du printf, le code d’insertion est également modifié, on utilise “%lc” au lieu de “%c”.

Sans cette méthode, l’affichage ressemble à ceci :



```
Veuillez saisir votre nombre:
4555
◆◆◆ bbb ◆◆◆ ◆◆◆
◆ bb ◆◆ ◆◆
```

Il faut également employer la fonction SETLOCALE. La fonction *setlocale()* est utilisée pour indiquer ou demander la localisation courante du programme.

Cette fonction est indispensable pour afficher l’unicode.

*setlocale(LC\_ALL, "")*

LC\_ALL: désigne toute la localisation

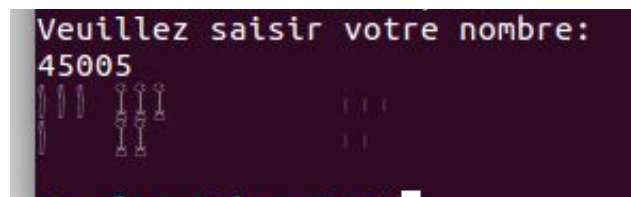
- Gestion du cadrage de l’affichage des nombres égyptiens

Suite à l'incorporation du tableau `decompose[ ]`, un nouveau problème d'affichage est apparu. En effet, lorsque l'utilisateur saisissait un nombre inférieur à 7 caractères, les cases du tableau non utilisés apparaissaient. Cela créait un espace inutile sur la gauche.

Pour remédier à ce problème, nous avons créé une fonction initialisation qui mesure le nombre de caractères du nombre saisi. Cette valeur est ensuite ajoutée au paramètre de la fonction d'affichage, ce qui va permettre à la fonction de savoir à partir de quel case l'impression des symboles doit commencer.

- Gestion du bug du 0

Le 0 n'étant pas représenté dans le système de numération égyptien, nous avons du adapter l'affichage des symboles. Par exemple, pour 7089 il y avait 3 espaces dans le terminal à la place du 0.



Nous avons donc ajouté un tableau qui permet de distinguer les zéros présents dans le nombre à convertir. En parcourant le tableau `decompose[ ]`, l'algorithme modifie la valeur de ce nouveau tableau. Pour 7089, `tab[0]=0`, `tab[1]=1`, `tab[2]=0` et `tab[3]=0`. On ajoute ensuite une condition, qui n'imprime rien si `tab[i]==1`.

### 3.4 Morceaux de code

Stockage des caractères unicode dans un tableau :

```
wchar_t *creationTabSigne()
{
    wchar_t *signe;
    signe = malloc(7 * sizeof(wchar_t));
    signe[0] = L'Ⲁ';
    signe[1] = L'ⲁ';
    signe[2] = L'Ⲃ';
    signe[3] = L'ⲃ';
    signe[4] = L'Ⲅ';
    signe[5] = L'ⲅ';
    signe[6] = L'Ⲇ';

    return (signe);
}
```

Afficher correctement le nombre égyptien en regroupant les mêmes signes :

```
int *decomposition(int nb)
{
    int *decompose;
    decompose = malloc(7 * sizeof(int));
    while (nb > 999999)
    {
        decompose[0]++;
        nb -= 1000000;
    }
    while (nb > 99999)
    {
        decompose[1]++;
        nb -= 100000;
    }
    while (nb > 9999)
    {
        decompose[2]++;
        nb -= 10000;
    }
}
```

Affichage des caractères unicode avec wchar:

```
for (j = 0; j < 3; j++)
    printf("%lc", signe[i - 1]);
decompose[7 - i] -= 3;
```

Résolution du problème du 0 dans l'affichage d'un nombre :

```
for (i = 0; i < 7; i++) // bug du 0
{
    if (decompose[i] == 0)
        tab[i] = 1;
}
for(cpt=0; cpt<3; cpt++)
{
    for (i = init; i > 0; i--)
    {
        if (tab[7 - i] == 1) // bug du 0
        {
        }
    }
}
```

Affichage horizontal du nombre dans le terminal (et non vertical sur une ligne)

```
else if (decompose[7 - i] > 3) // Cas 1
{
    for (j = 0; j < 3; j++)
        printf("%lc", signe[i - 1]);
    decompose[7 - i] -= 3;
}
else if (decompose[7 - i] > 0) // Cas 2
{
    for (j = 0; j < decompose[7 - i]; j++)
        printf("%lc", signe[i - 1]);
    for (j = 0; j < 3 - decompose[7 - i]; j++)
        printf(" ");
    decompose[7 - i] = 0;
}
else // Cas 3
    printf(" ");
if (tab[8 - i] != 1)
    printf(" ");
```

Affichage de l'unicode dans tout les terminals:

```
#include <locale.h> //au debut du code  
  
setlocale(LC_ALL, ""); //dans le main
```



## CONCLUSION

### 4.1 Technique

Au terme de ce projet, nous pouvons affirmer que notre programme C de conversion des nombres égyptiens semble compiler parfaitement et ne présente pas d'erreur de codage, ni de bugs.

Les symboles égyptiens s'affichent correctement grâce au package installé préalablement.

L'utilisateur munit du manuel d'utilisation ne devrait pas avoir de problèmes pour compiler et lancer le code.

### 4.2 Générale

Pour conclure notre programme effectue bien une conversion de chiffre décimal dans le système numérique égyptien et respecte le "code", la manière qu'avaient les égyptiens d'écrire leurs nombres.

### 4.3 Ouverture possible

Pour aller plus loin dans ce projet, nous pourrions modifier notre programme pour avoir une interface graphique plus "user-friendly" et ajouter la possibilité d'effectuer les opérations de base (additions, soustractions, divisions euclidienne....).

## ANNEXE

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <wchar.h>
4  #include <locale.h>
5  #include "convertisseurEgyptien.h"
6
7  int initialisation(int *decompose)
8  {
9      int i;
10
11      for(i = 0; i < 7; i++)
12      {
13          if (decompose[i] != 0)
14              return (7 - i);
15      }
16      return 0;
17  }
18
19
20 wchar_t *creationTabSigne()
21 {
22     wchar_t *signe;
23     signe = malloc(7 * sizeof(wchar_t));
24     signe[0] = L'';
25     signe[1] = L'';
26     signe[2] = L'';
27     signe[3] = L'';
28     signe[4] = L'';
29     signe[5] = L'';
30     signe[6] = L'';
31
32     return (signe);
33 }
34
35 int *decomposition(int nb)
36 {
37     int *decompose;
38     decompose = malloc(7 * sizeof(int));
39     while (nb > 999999)
40     {
41         decompose[0]++;
42         nb -= 1000000;
43     }
44     while (nb > 99999)
45     {
46         decompose[1]++;
47         nb -= 100000;
48     }
49     while (nb > 9999)
50     {
51         decompose[2]++;
52         nb -= 10000;
53     }
54     while (nb > 999)
55     {
56         decompose[3]++;
57         nb -= 1000;
58     }
59     while (nb > 99)
60     {
61         decompose[4]++;
62         nb -= 100;
63     }
64     while (nb > 9)
65     {
66         decompose[5]++;
67         nb -= 10;
68     }
69     while (nb > 0)
```

```

69     while (nb > 0)
70     {
71         decompose[6]++;
72         nb -= 1;
73     }
74
75     return (decompose);
76
77 }
78
79 void affichage(int *decompose, wchar_t *signe, int init)
80 {
81     int i;
82     int j;
83     int cpt;
84     int tab[7] = {0};
85
86     for (i = 0; i < 7; i++) // bug du 0
87     {
88         if (decompose[i] == 0)
89             tab[i] = 1;
90     }
91     for(cpt=0; cpt<3; cpt++)
92     {
93         for (i = init; i > 0; i--)
94         {
95             if (tab[7 - i] == 1) // bug du 0
96             {
97             }
98             else if (decompose[7 - i] > 3) // Cas 1
99             {
100                 for (j = 0; j < 3; j++)
101                     printf("%lc", signe[i - 1]);
102                 decompose[7 - i] -= 3;
103             }
104
105             else if (decompose[7 - i] > 0) // Cas 2
106             {
107                 for (j = 0; j < decompose[7 - i]; j++)
108                     printf("%lc", signe[i - 1]);
109                 for (j = 0; j < 3 - decompose[7 - i]; j++)
110                     printf(" ");
111                 decompose[7 - i] = 0;
112             }
113             else // Cas 3
114                 printf(" ");
115             if (tab[8 - i] != 1)
116                 printf(" ");
117         }
118         printf("\n");
119     }
120
121 }
122
123
124 int main(void)
125 {
126     int nb;
127     int *decompose;
128
129     //declaration de la variable qui indique ou commence le nombre
130     int init = 0;
131     int *signe;
132
133     setlocale(LC_ALL, "");
134     signe = creationTabSigne();
135     printf("Veuillez saisir votre nombre:\n");
136     scanf("%d", &nb);
137     //decomposition du nombre saisi
138     decompose = decomposition(nb);
139     init = initialisation(decompose);
140     affichage(decompose, signe, init);

```

## WEBOGRAPHIE

- <https://fr.wikipedia.org/wiki/Aide:Unicode#GNU/Linux>
- <http://www.clempinch.com/convertisseur/>
- [https://fr.wikipedia.org/wiki/%C3%89gypte\\_antique](https://fr.wikipedia.org/wiki/%C3%89gypte_antique)
- <https://www.dcode.fr/conversion-base-n>
- [http://www.math.univ-angers.fr/~labatte/l3sen/Cours/TD\\_NUMERATION.pdf](http://www.math.univ-angers.fr/~labatte/l3sen/Cours/TD_NUMERATION.pdf)
- <http://matoumatheux.ac-rennes.fr/num/numeration/babylonien.htm>
- <https://www.dcode.fr/nombres-babyloniens>